

User Manual

for

DA11000-12-4M-PCI DA11000-12-16M-PCI

1 Channel, 1.0 GSPS, 12-Bit, PCI Arbitrary Waveform Generator Card

Last Modified 5/30/2009

CHASE SCIENTIFIC COMPANY

P.O. Box 1487
Langley, WA 98260

Tel: 360-221-8455

Fax: 360-221-8457

Email: techsupport@chase2000.com

Web: <http://www.chase2000.com>

© Copyright 2004-2009 by
Chase Scientific Company

This manual, the DA11000 card, and the software drivers outlined in this document are copyrighted with all rights reserved. Under the copyright laws, the above mentioned may not be copied, in whole or in part, without the express written consent of Chase Scientific Company.

TABLE OF CONTENTS

1 GENERAL INFORMATION..... 4

1.1 INTRODUCTION..... 4

1.2 REFERENCES..... 4

1.3 DELIVERABLES..... 5

 1.3.1 Software..... 5

 1.3.2 Hardware..... 5

 1.3.3 Checklist..... 6

1.4 PRODUCT SPECIFICATION..... 6

1.5 OPTION SUMMARY..... 8

1.6 TECHNICAL SUPPORT / SOFTWARE UPDATES..... 8

1.7 WARRANTY..... 9

2 HARDWARE DESCRIPTION.....10

2.1 INTRODUCTION..... 10

2.2 BLOCK DIAGRAM..... 10

2.3 BOARD DRAWING..... 11

2.4 EXTERNAL CLOCK JUMPER CONFIGURATIONS..... 12

2.5 PCI MEMORY ALLOCATION..... 12

3 THEORY OF OPERATION..... 13

3.1 INTRODUCTION..... 13

3.2 DOWNLOADING AND OUTPUTTING USER DATA TO THE DA11000..... 13

4 SOFTWARE DRIVERS..... 13

4.1 INTRODUCTION..... 14

4.2 DRIVER INSTALLATION..... 14

 4.2.1 Windows 98 / ME / NT4..... 14

 4.2.2 Windows 2000 / XP..... 14

4.3 FUNCTION CALLS..... 15

 4.3.1 C Header File for DLL..... 15

 4.3.2 Function Call Descriptions / Usage..... 16

 4.3.2.1 da11000_CountCards()..... 16

 4.3.2.2 da11000_Open()..... 16

 4.3.2.3 da11000_Close()..... 17

 4.3.2.4 da11000_SetClock()..... 17

 4.3.2.5 da11000_SetTriggerMode()..... 17

 4.3.2.6 da11000_SetSoftTrigger()..... 18

 4.3.2.7 da11000_SetMarkers()..... 18

 4.3.2.8 da11000_SetOffset() [Not Available at this time]..... 19

 4.3.2.9 da11000_CreateSingleSegment()..... 20

 4.3.2.10 da11000_CreateSegments()..... 20

 4.3.2.11 da11000_Set_Atten() [only works for DA11000 with option 1]..... 22

 4.3.2.12 da11000_UpdateSegmentCmds ()..... 22

4.4 PROGRAMMING EXAMPLES..... 23

 4.4.1 Using Windows 95/98/NT DLL..... 23

5 MISCELLANEOUS..... 25

5.1 CALIBRATION..... 25

5.2 MAINTANENCE..... 25

5.3 CHANGES/CORRECTIONS TO THIS MANUAL..... 25

ILLUSTRATIONS / TABLES

FIGURE 1 – BLOCK DIAGRAM..... 10
FIGURE 2 – BOARD LAYOUT..... 11

“da11000_manual.odt” was created on 6/9/04 and last modified on 5/30/2009

1 GENERAL INFORMATION

1.1 Introduction

The DA11000 is a (1) Channel, 12-bit, 1.0 GigaSample/second Arbitrary Waveform Generator on a single mid-sized PCI card. It comes standard with following general features:

- Standard Fixed Clock 1.0 GSPS
- (1) Channel Analog Output, 12-bit Vertical Resolution
- (1) TTL Output Marker
- Programmable Segment Size from 64 Words to full memory
- Programmable Number of Segments up to 32K
- External TTL clock and External TTL trigger input

The analog output consist of (1) 50 ohm SMA output. To provide maximum flexilbilty and performance to the user, the outputs come unfiltered. An appropriate low pass filter is generally added in-line for a particular application and can be bought from companies like Mini-Circuits or can be ordered and/or custom made directly from Chase Scientific.

The DA11000 has TTL input triggering capability that allows a segment or segments of data to be output only after a trigger is present. Gating is also available which will start and stop the data from being output on high or low TTL levels respectively.

1.2 References

PCI Local Bus Specification, Rev. 2.1, June 1st, 1995. For more information on this document contact:
PCI Special Interest Group
P.O Box 14070
Portland, OR 97214

Phone (800) 433-5177 (U.S.)
(503) 797-4207 (International)
FAX (503) 234-6762

1.3 Deliverables

1.3.1 Software

The DA11000 comes with DLL drivers for **Windows 98/ME/NT4/2000/XP**. Software comes on a single 3.5" diskette (or CDR disk). Call Chase Scientific for the latest information on drivers for other operating system platforms.

Windows drivers are provided as a Dynamic Link Library (*.DLL) which is compatible with most 32-bit windows based development software including Microsoft C/C++, Borland C/C++, and Borland Delphi. This DLL uses the "cdecl" calling convention for maximum compatibility and was made using Borland C++ Builder. It automatically provides the interface to the system drivers "Windrvr6.sys" for Windows 98/ME/NT4/2000/XP.

Actual Listing of files on Diskette:

```

-----  D I R E C T O R I E S / F I L E S  -----
BASE_DIR
|
|  readme.txt                // This file.
|
|  dal1000_manual.pdf        // Manual for DA11000 in PDF format
|  dal1000_ref_drwg.pdf      // Reference Drawing (Connector Descriptions)
|
|  Register_DA11000_Win2000_XP.bat // Installs Kernel driver for Win2000/XP
|  UnRegister_DA11000_Win2000_XP.bat // Uninstalls Kernel driver for Win2000/XP
|
|  Register_DA11000_Win98_ME_NT4.bat // Installs Kernel driver for Win98/ME/NT4.0
|  UnRegister_DA11000_Win98_ME_NT4.bat // Uninstalls Kernel driver for Win98/ME/NT4.0
|
|  wdregl6.exe              // Called by Register_DA11000_Win98_ME_NT4.bat
|  wdreg.exe                // Called by Register_DA11000_Win2000_XP.bat
|  windrvr6.inf            // Setup information file automatically called by above exe(s).
|
|  dal1000_dll.dll          // DLL for 98/ME/NT4/2000/XP ( extern "C" __declspec(dllexport) )
|  dal1000_dll_import.h    // Header file for DLL
|  dal1000_dll.lib         // Library file for DLL in Borland C++
|
|  | MS_LIB File
|  | | dal1000_dll.lib      // Include in MSVC Project to compile DLL above
|  | | dal1000_dll_import.h // Header file for DLL
|
|  dal1000.exe              // Simple GUI to test DLL and Kernel drivers
|
|  Chase_DA11000.inf       // Plug-And-Play file needed by 98/ME/NT4/2000/XP for automatic
|                          // hardware configuration.
|
|  windrvr6.sys           // Windows 98/ME/NT4/2000/XP Driver - copy this virtual driver
|                          // to "c:\<windir>\system32\drivers\" if not automatically done
|                          // so after running batch file.
|
|  | Sample Waveform Files
|  | | 64bit_sqwv.txt      // 64 sample squarewave full scale
|  | | 64K_Data.txt       // 64K sample of lorentzian pulses (disk drive)
|  | | random_noise.txt   // Random noise
|
-----  E N D  -----

```

1.3.2 Hardware

The DA11000 hardware consists of a single mid-sized PCI compliant card. The card is shipped with this manual which includes complete hardware and software descriptions.

1.3.3 Checklist

| Item # | Qty | Part Number | Description |
|--------|-----|-------------------|---|
| 1 | 1 | DA11000-12-1M-PCI | 1.0 GSPS, Arbitrary Waveform Generator, PCI card. |
| 2 | 1 | DA11000 Drivers | 3.5" diskette (or Mini-CDR) with Dynamic Link Libraries for Windows 95/98/ME/NT4/2000/XP. Includes examples and manual. |

1.4 Product Specification

(all specifications are at 25 °C unless otherwise specified)

SPECIFICATIONS

| Parameter | Conditions | Typical Values unless otherwise indicated |
|--------------------------------|--------------|--|
| Analog Outputs | | |
| Number of Outputs | | (1) 50 ohm SMA outputs |
| Output Coupling | | AC coupling through 50 ohm transformer |
| Vertical Resolution | | 12 bits (1 part in 4096) |
| Amplitude | 1.0 GS/s | 0.5Vpp +/-3%, single ended into 50 ohms. |
| Programmable Attenuator | | |
| | | (Option 1) |
| Attenuation Range | | 31.5 dB |
| Number of Steps | | 64 steps |
| Attenuation per Step | | 0.5 dB (typical) |
| Insertion Loss | | 1.3 dB (typical at 100 MHz) |
| Rise Time (10% to 90%) | No Filters | 300 psec typical into 50 ohms |
| Fall Time (10% to 90%) | No Filters | 300 psec typical into 50 ohms |
| Clock Jitter | 1.0 GS/sec | Less than 20 psec RMS at 300MHz |
| Trigger Delay | 1.0 GS/sec | TBD |
| SFDR | | |
| Fout < 200MHz | 1.0 GS/sec | < -50 dB Typical |
| Fout = 200-400 MHz | 1.0 GS/sec | TBD |
| Internal Clock Rate | | |
| Frequency Range | | 1.0 GHz Fixed |
| Resolution | | N/A |
| Stability | T=0°C – 70°C | 100 ppm |
| Memory | | |
| Waveform | Standard | 4 MWords x 12-bits (Optional 16M) |
| # of User Segments | | 1 to 16K segments |
| Segment Size Range | | 64 Words up to total memory in 64 Word increments (minus pads) |
| Digital Outputs | | |
| (1) TTL Markers | | Fclk/4 resolution |
| Digital Inputs | | |

| | | |
|-----------------------------|-----------------------|--|
| High Speed Clk input | | 50 ohm SMA input Can only use the following frequencies: 1.0 GHz, 500 MHz, 250 MHz, and 125 MHz. |
| TTL Trigger input | | Used to initiate any memory segment programmed for that purpose. |
| External Clock Input | | |
| High Speed Clk input | D/A rate | 50 ohm SMA input (Sinewave from 0dBm to 6 dBm) Can only use the following frequencies: 1.0 GHz, 500 MHz, 250 MHz, and 125 MHz. (See section 2.4 for jumper settings) |
| Low Speed Clk input | 10 MHz Reference Only | 50 ohm SMA input (Sine/Square wave from 0dBm to 10 dBm) (See section 2.4 for jumper settings) |

ENVIRONMENTAL

| Parameter | Typical Values unless otherwise stated |
|---------------|---|
| Temperature | |
| Operating | 0 to 70 degrees C standard |
| Non-Operating | -40 to +85 degrees C extended |
| Humidity | 5 to 95% non-condensing |
| Operating | 20% to 80% |
| Non-Operating | 5% to 95% |
| Power | |
| +5V | +5V DC +/- 10% => 500mA, 2.5 Watts (Typical using worst case waveform) |
| +3.3V | +3.3 VDC +/- 10% => 2.5 Amps, 8.4 Watts (Typical using worst case waveform) |
| +12V | +12 VDC +/- 10% => 216mA, 2.6 Watts (Typical using worst case waveform) |
| -12V | -12 VDC +/- 10% => 100mA, 1.2 Watts (Typical using worst case waveform) |
| Size | |
| Basic DA11000 | (1) Mid-Sized PCI Card |

1.5 Option Summary

OPTION SUMMARY

| Option Name | Description |
|-------------|-------------------------|
| Option 3 | Programmable Attenuator |
| Option 5 | Linux Drivers |

1.6 Technical Support / Software Updates

For technical support:

| | |
|--------|--|
| Phone: | 360-221-8455 |
| Fax: | 360-221-8457 |
| Email: | techsupport@chase2000.com |
| Mail: | Chase Scientific Company P.O. Box 1487 Langley, WA 98260 |

For software updates:

| | |
|--------|--|
| Email: | techsupport@chase2000.com |
| Web: | http://www.chase2000.com |

1.7 Warranty

Chase Scientific Company (hereafter called Chase Scientific) warrants to the original purchaser that its DA11000, and the component parts thereof, will be free from defects in workmanship and materials for a period of ONE YEAR from the date of purchase.

Chase Scientific will, without charge, repair or replace at its option, defective or component parts upon delivery to Chase Scientific's service department within the warranty period accompanied by proof of purchase date in the form of a sales receipt.

EXCLUSIONS: This warranty does not apply in the event of misuse or abuse of the product or as a result of unauthorized alterations or repairs. It is void if the serial number is altered, defaced or removed.

Chase Scientific shall not be liable for any consequential damages, including without limitation damages resulting from loss of use. Some states do not allow limitation or incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific rights. You may also have other rights that vary from state to state.

Chase Scientific warrants products sold only in the USA and Canada. In countries other than the USA, each distributor warrants the Chase Scientific products that it sells.

NOTICE: Chase Scientific reserves the right to make changes and/or improvements in the product(s) described in this manual at any time without notice.

2 HARDWARE DESCRIPTION

2.1 Introduction

The DA11000 hardware consists of the following major connections:

- (1) Normal, 1.0Gigasamples/second, 12-bit analog output (SMA)
- (1) PECL/Sinewave Clock Input, 125MHz, 250MHz, 500MHz, 1.0 GHz ONLY (AC coupled)
- (1) TTL Trigger input (SMA)
- (1) TTL Outputs Markers (SMA)

2.2 Block Diagram

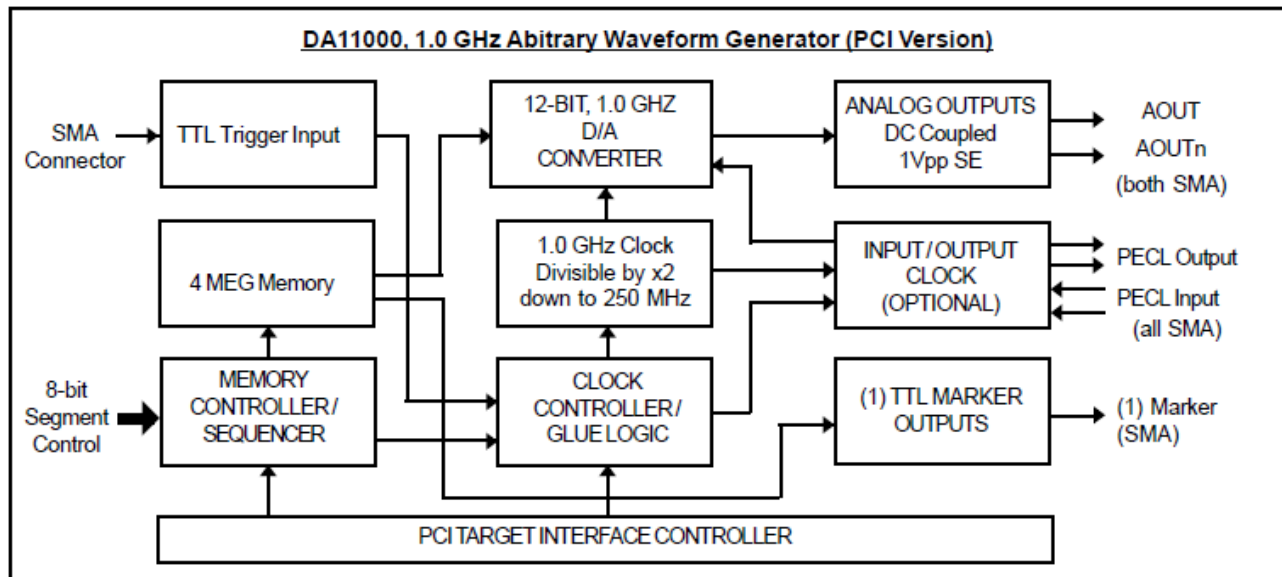


Figure 1 – Block Diagram

2.3 Board Drawing

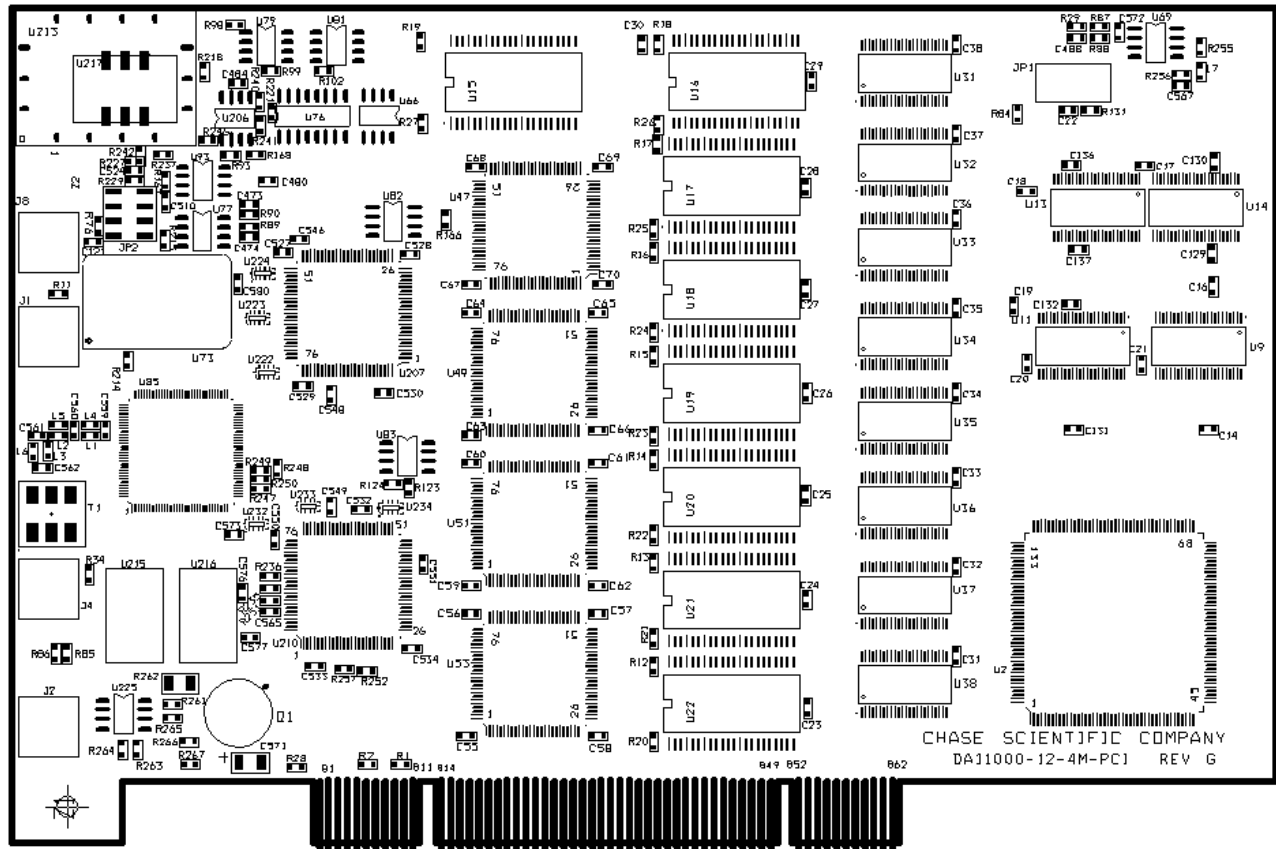


Figure 2 – Board Layout

2.4 External Clock Jumper Configurations

| | | |
|---|---|------------------------|
| 1 | 2 | <i>JUMPER 1</i> |
| 3 | 4 | <i>JUMPER 2</i> |
| 5 | 6 | <i>JUMPER 3</i> |
| 7 | 8 | <i>JUMPER 4</i> |

Clock Configuration Table

| | Jumper #1 | Jumper #2 | Jumper #3 | Jumper #4 |
|---|-----------|-----------|-----------|-----------|
| Use 1.0 GHz Internal Clock locked to internal 10 MHz reference | SHORT | OPEN | OPEN | SHORT |
| Use External Clock at D/A clock rate Allowable rates = 1.0 GHz, 500 MHz, 250 MHz | OPEN | SHORT | OPEN | SHORT |
| Lock to External 10 MHz Reference DA11000 Phase locks the internal 1.0 GHz master clock to external 10 MHz Reference | SHORT | OPEN | SHORT | OPEN |

| External Clock Input Notes | | |
|----------------------------|-----------------------|--|
| High Speed Clk input | D/A rate | 50 ohm SMA input (Sinewave from 0dBm to 6 dBm) Can only use the following frequencies: 1.0 GHz, 500 MHz, 250 MHz, and 125 MHz. |
| Low Speed Clk input | 10 MHz Reference Only | 50 ohm SMA input (Sine/Square wave from 0dBm to 10 dBm) |

2.5 PCI Memory Allocation

DA11000 on-board memory is mapped automatically when a PCI 2.1 (or newer) motherboard powers up. If the DA11000 has 4 MegaSamples of memory, then the motherboard will allocate 8 Megabytes of memory. Once installed, the DA11000 software drivers will find the board or boards without the user changing any jumpers or worrying about addressing. Unless the user manages to use up the entire memory available to the PCI motherboard (usually 256 Megabytes or more), then how the memory is allocated and where it is located in the system is completely transparent.

3 THEORY OF OPERATION

3.1 Introduction

Although the DA11000 is primarily comprised of a **Segment Sequencer** (or memory manager) and a 4:1 **High Speed Multiplexor**, it's how the software interacts with the hardware that makes it work. The following sections should provide enough operational theory for better understanding when using the software drivers.

3.2 Downloading and Outputting User Data to the DA11000

The DA11000 RAM memory IC's not only contain the user's waveform data, but it also contains special command codes that run the Segment Sequencer. These codes are placed into the upper nibble (4 bits) of selected individual sample points (16 bit words), leaving the lower 12 bits for user data. The Segment Sequencer reads these codes to determine where and when to jump to another segment, how many times to loop, when to wait for a trigger, and when to shut down. This is the heart of the DA11000 memory management.

Downloading a Single User Waveform (single segment) into memory is performed by simply calling `da11000_CreateSingleSegment(DWORD CardNum, DWORD NumPoints, DWORD NumLoops, PVOID UserArrayPtr, DWORD TrigEn)`. The user must be sure to pass the size of the waveform (`NumPoints`), the number of times to repeat the waveform (`NumLoops`), a pointer variable pointing to the user array containing the data (`UserArrayPtr`), and finally, whether the segment will be self triggered or triggered by an external signal (`TrigEn`).

Downloading Multiple Linked Waveform Segments is performed by calling `da11000_CreateSegments(DWORD CardNum, DWORD NumSegments, PVOID PtrToSegmentsList)`. This function call requires the user to create a structure containing all the critical information on the segments that the user wants to download. The actual structure for each segment looks like the following:

```
typedef struct
{
    DWORD    SegmentNum;    // Current Segment Number
    PVOID    SegmentPtr;   // Pointer to current user segment
                                // ==> elements of one dimensional array must
                                // be of type WORD
    DWORD    NumPoints;    // Number of points in segment (must be multiple of 16)
    DWORD    NumLoops;     // Number of times to repeat segment (applies
                                // to next segment)
    DWORD    BeginPadVal;  // Pad value for beginning of triggered segment
    DWORD    EndingPadVal; // Pad value for ending of triggered segment
    DWORD    TrigEn;       // If > 0 then wait for trigger before going to
                                // next segment.
    DWORD    NextSegNum;   // Next segment to jump to after completion
                                // of current segment activities
} SegmentStruct;
```

The user must create an array of these segments and pass the pointer (`PtrToSegmentsList`) to the function call.

After the appropriate waveform data has been downloaded to the DA11000, `da11000_SetTriggerMode()` is enabled and the output begins.

4 SOFTWARE DRIVERS

4.1 Introduction

Our primary objective in designing software drivers is to get the user up and running as quickly as possible. While the details on individual function calls are listed in sections 4.3.x, the programming examples in section 4.4.x will show you how to include them into your programs. Please note that function calls are the same whether you are calling them under Windows 98, ME, NT4, 2000, or XP.

4.2 Driver Installation

4.2.1 Windows 98 / ME / NT4

- 1) Do not install DA11000 card at this time.
- 2) UnZip all files into directory "C:\temp\da11000\" (create directories if needed) You can move and/or copy the files later to a directory of your choice.
- 3) Run da11000_Register_Win98_ME_NT4.bat. This will copy the Kernel driver windrvr6.sys to "c:\<windir>\system32\drivers\" directory and will register the Kernel driver in the Windows Registry so that it starts up each time the computer is rebooted.
- 4) Power off computer. Insert DA11000 card. Power up computer.
- 5) When OS asks for Driver File point to "Chase_DA11000.inf". If OS does not ask for file, then check hardware configuration and update if not listed properly under "Jungo" in Device Manager (see below).

To check to see which driver is installed, do the following:

=> Control Panel
=> System
=> Hardware
=> Device Manager
=> **Jungo**
=> **Chase_DA11000** (Both this and WinDriver below should be present)
=> **WinDriver**

If you see another driver in place of "Chase_DA11000", then right click the first device under Jungo and click properties. Update the driver by pointing to "Chase_DA11000". You may have to go through a series of menus.

4.2.2 Windows 2000 / XP

- 1) Do not install DA11000 card at this time.
- 2) UnZip all files into directory "C:\temp\da11000\" (create directories if needed) You can move and/or copy the files later to a directory of your choice.

- 3) Run da11000_Register_Win2000_XP.bat. This will copy the Kernel driver windrvr6.sys to "c:\<windir>\system32\drivers\" directory and will register the Kernel driver in the Windows Registry so that it starts up each time the computer is rebooted.
- 4) Power off computer. Insert DA11000 card. Power up computer.
- 5) When OS asks for Driver File point to "Chase_DA11000.inf". If OS does not ask for file, then check hardware configuration and update if not listed properly under "Jungo" in Device Manager (see below).

To check to see which driver is installed, do the following:

```

=> Control Panel
  => System
    => Hardware
      => Device Manager
        => Jungo
          Chase_DA11000 (Both this and WinDriver below should be present)
          WinDriver

```

If you see another driver in place of "Chase_DA11000", then right click the first device under Jungo and click properties. Update the driver by pointing to "Chase_DA11000". You may have to go through a series of menus.

4.3 Function Calls

4.3.1 C Header File for DLL

```

//-----
//  USER ROUTINES
//-----

#define IMPORT extern "C" __declspec(dllimport)

IMPORT DWORD da11000_CountCards(void);
IMPORT DWORD da11000_Open(DWORD CardNum);
IMPORT DWORD da11000_Close(DWORD CardNum);

IMPORT void da11000_SetClock(DWORD CardNum, DWORD Frequency);

IMPORT void da11000_SetTriggerMode(DWORD CardNum, BYTE Mode, BYTE ExtPol);
IMPORT void da11000_SetSoftTrigger(DWORD CardNum);
IMPORT void da11000_SetMarkers(DWORD CardNum, DWORD PointAddr, BYTE Nib1, BYTE Nib2);
IMPORT void da11000_SetOffset(DWORD CardNum, int Mode, int Offset);

IMPORT void da11000_CreateSingleSegment(DWORD CardNum, DWORD ChanNum, DWORD NumPoints,
                                         DWORD NumLoops, PVOID UserArrayPtr, DWORD TrigEn);
IMPORT void da11000_CreateSegments(DWORD CardNum, DWORD NumSegments, PVOID UserSegmentsPtr);
IMPORT void da11000_UpdateSegmentCmds(DWORD CardNum, DWORD NumSegments, PVOID PtrToSegmentsList);

```

4.3.2 Function Call Descriptions / Usage

4.3.2.1 da11000_CountCards()

Description

Returns number of DA11000 cards present on computer.

Declaration

```
DWORD da11000_CountCards(void);
```

Parameters

none

Return Value

Returns with an encoded value which represents the number of DA11000.

Return Values:

- 1-4: Number of DA11000 boards detected.
- 0: Indicates that no boards were found but that drivers are working properly.
- 13: Software drivers are not installed properly.
working correctly. "13"

Example

```
DWORD Num_da11000_Boards = da11000_Open() & 0x3;
```

4.3.2.2 da11000_Open()

Description

Loads the DA11000 software drivers and sets the DA11000 board to its default state.

Declaration

```
DWORD da11000_Open(DWORD CardNum);
```

Parameters

CardNum: 1 <= CardNum <= 4

Return Value

Returns with error code. A "0" means everything is fine. See below for details for other values.

Return Values:

- 0: Opened Windriver Successfully and DA11000 Card Found Successfully
- 1: Opened Windriver Successfully, but NO DA11000 CARDS FOUND
- 2: Opened Windriver Successfully, Card found, but unable to open.
- 3: Opened Windriver Successfully, Board already open.
- 6: Card number exceeds number of cards.
- 13: FAILED TO OPEN Windriver Kernel Driver

Example

```
DWORD OpenErrorCode = da11000_Open(1); // Opens Board Number 1 and stores value.
```

4.3.2.3 da11000_Close()

Description

Closes DA11000 drivers. Should be called after finishing using the driver. However, if no other software uses the “windrv.xxx” (usual situation), then there is no need to close it until user is ready to completely exit from using their main software program which calls “windrv.xxx”. If the user is loading the “windrv.xxx” dynamically (during run time), then they should close before unloading the driver.

Declaration

```
DWORD da11000_Close(DWORD CardNum);
```

Parameters

CardNum: 1 <= CardNum <= 4

Return Value

Returns with error code. A "0" means everything is fine. See below for details for other values.

Return Values:

- 0: Closed Windriver Successfully for DA11000 card requested.
- 5: DA11000 Card Already Closed for card requested.
- 13: FAILED TO ACCESS Windriver Kernel Driver

Example

```
DWORD CloseErrorCode = da11000_Close(1);
```

4.3.2.4 da11000_SetClock()

Description

Sets the Digital to Analog converter clock rate. This function does nothing (placeholder) at this time on the DA11000. The card is fixed at 1.0 GSPS. There are jumpers on the PCB to allow for external clock features (see section 2.4).

Declaration

```
void da11000_SetClock(DWORD CardNum, DWORD Frequency);
```

Parameters

CardNum: 1 <= CardNum <= 4
Frequency: 1000000000

Return Value

none

Example

```
da11000_SetClock(300000000); // Sets clock rate to 300 MHz.
```

4.3.2.5 da11000_SetTriggerMode()

Description

Sets triggering modes. This command should be called (using mode=0) just after the driver is opened to initialize internal hardware registers before calling any other routines. This function also selects whether board is in triggered mode or not and polarity of external TTL triggered signal.

Declaration

```
void da11000_SetTriggerMode(DWORD CardNum, BYTE Mode, BYTE ExtPol);
```

Parameters

CardNum: 1 <= CardNum <= 4

Mode:

- 0: Shuts down all output operations. Asynchronously resets RAM address counter and repeat counters to zero.
- 1: Used for starting single segment operation for segment created with “da11000_CreateSingleSegment()”. Repeats indefinitely until mode set back to 0. External or “soft” trigger has no effect in this mode. Also works for “da11000_CreateSegments()”, but any segments specified as triggered will immediately jump to next segment (no trigger required) and beginning and ending pads will be present in output for these segments.
- 2: Sets up first segment for external or “soft” trigger mode. Individual segment(s) created as triggered will wait until external or soft trigger has occurred. If segment was created not to be triggered, then segment will follow previous segment in a continuous fashion (no trigger needed). See da11000_CreateSegments for more information on multi-segment functioning.

ExtPol:

- 0: Trigger is initiated on RISING edge of TTL waveform.
- 1: Trigger is initiated on FALLING edge of TTL waveform.

Return Value

none

Example

```
da11000_SetTriggerMode(2,0); // First segment will wait for trigger before
                             // running.
```

4.3.2.6 da11000_SetSoftTrigger()

Description

Emulates external triggering in software. Since this function actually toggles polarity of external input signal, it is “ORed” with external signal.

Declaration

```
void da11000_SetSoftTrigger(DWORD CardNum);
```

Parameters

none

Return Value

none

Example

```
da11000_SetSoftTrigger(1); // Initiates software trigger on Card Number 1
```

4.3.2.7 da11000_SetMarkers()

Description

Sets up TTL output marker locations relative to waveform memory. It is up to the user to place the markers correctly. There is always a startup 64 sample leading pad when a waveform first outputs. Please note that all segments have a 64 sample leading pad and a 64 sample trailing pad, regardless of whether they repeat or not.

Resolution of the markers is 1/4 of the clock rate. Also, please note that this function call must be called after creating any segments since `da11000_CreateSegments()` and `da11000_CreateSingleSegment()` will overwrite the markers with zeros if done in the reverse order.

Declaration

```
void da11000_SetMarkers(DWORD CardNum, DWORD PointAddr, BYTE Nib1, BYTE Nib2);
```

Parameters

CardNum: 1 <= CardNum <= 4

PointAddr: RAM address location. Minimum resolution is 4 clock samples.

Nib1: 0 <= Nib1 <= 0xF [see board layout for connector information]

Nib2: 0 <= Nib2 <= 0xF

Return Value

None.

Example

```
da11000_SetMarkers(1, 64, 0xF, 0xF); // Place marker on all bits at
// beginning of 1st data segment of
// board number 1.
```

4.3.2.8 da11000_SetOffset() [Not Available at this time]**Description**

Sets output voltage offsets on Normal and Complementary outputs on primary DA11000 board.

Declaration

```
void da11000_SetOffset(DWORD CardNum, int Mode, int Offset);
```

Parameters

CardNum: 1 <= CardNum <= 4

Mode:

0 = Adjust offset on Complimentary Output

1 = Adjust offset on Normal Output

2 = Adjust both at same time

Offset: -250 <= Offset <= 250 [Resolution = 1; Units are in millivolts DC]

Return Value

none

Example

```
da11000_SetOffset(1,2,22); // Sets both output DC offset to 22mV for card 1.
```

4.3.2.9 da11000_CreateSingleSegment()

Description

Creates a single segment in memory. The user determines the size of the array and whether the segment is started automatically or waits for an external input trigger. After creating a single segment waveform, the user must call SetTriggerMode() to turn on/off output waveforms.

In triggered mode there is a 64 samples of pad at the beginning and end of the segment with a level set at 2047. In non-triggered mode the only pad that is visible is the beginning pad when the output is started, then repeats data portion indefinitely until reset. All segments, regardless of whether it's triggered or not, have 64 sample pads at the beginning and end of the segments in actual memory, but may not be visible depending on whether the segment is triggered or not. See "da11000_CreateSegments()" for generating multiplied segments.

Declaration

```
void da11000_CreateSingleSegment(DWORD CardNum,
                                DWORD ChanNum,
                                DWORD NumPoints,
                                DWORD NumLoops,
                                PVOID UserArrayPtr,
                                DWORD TrigEn);
```

Parameters

CardNum: 1 <= CardNum <= 4
 ChanNum: 0x01, 0x02, 0x04, 0x08 for channels 1,2,3, and 4 [DA11000]
 NumPoints: 0 <= NumPoints <= (MaxMem-128) [*** MUST BE MULTIPLE OF 64 ***]
 NumLoops: Set to 0 (other values not available) [0 = Continuous]
 UserArrayPtr: Pointer to user array of WORD
 TrigEn: High enables external trigger (must also set da11000_SetTriggerMode to triggered)

Return Value

None.

Example

```
da11000_CreateSingleSegment(1, // Card Number 1
                            1, // Channel 1
                            128, // 128 Words contained
                            0, // Loops continuously
                            UserArrayPointer, // Pointer to user data
                            0); // External trigger not enabled
```

4.3.2.10 da11000_CreateSegments()

Description

Creates any number of segments up to the size of memory. All segments have 64 samples of beginning pad and 64 samples of trailing pad which the user cannot access except to determine the default levels. However, when repeating or jumping in non-triggered mode, the user will not see the pad fields. Each segment can be programmed for repeat counts up to 16K and can jump to any other segment. See below for data structures for creating user segments. User must provide the correct array structures and pass a pointer to it along with how many sequential segments are desired to be used.

After creating a complete waveform, the user must call SetTriggerMode() to turn on/off output waveforms.

Declaration

```
void da11000_CreateSegments(DWORD CardNum,
                           DWORD ChanNum,
                           DWORD NumSegments,
                           PVOID PtrToSegmentsList);
```

Parameters

```
CardNum:      1 <= CardNum <= 4
ChanNum:      0x01 (only valid number)
```

```
NumSegments:  Number of segment structures (see below) which user has
                defined and wants to use.
```

```
PtrToSegmentsList: Pointer to user array with each element with structure
                    defined as shown below.
```

```
typedef struct
{
    DWORD    SegmentNum;    // Current Segment Number
    PVOID    SegmentPtr;    // Pointer to current user segment
                                // ==> elements of one dimensional array must
                                // be of type WORD
    DWORD    NumPoints;     // Number of points in segment
                                // [ *** MUST BE MULTIPLE OF 64 *** ]

    DWORD    NumLoops;     // Number of times to repeat segment (applies
                                // to next segment)
    DWORD    BeginPadVal;   // Pad value for beginning of triggered segment
    DWORD    EndingPadVal;  // Pad value for ending of triggered segment
    DWORD    TrigEn;        // If > 0 then wait for trigger before going to
                                // next segment.
    DWORD    NextSegNum;    // Next segment to jump to after completion
                                // of current segment activities
} SegmentStruct;
```

**** Note that a segment is determined to be a triggered segment by the previous segment. So setting Segment 5 as triggered will stop the sequence after Segment 5 has executed and will wait for trigger event before "NextSegNum" is started.

The first segment is a special case and is determined by default as a triggered type if SetTriggerMode() is set to "mode=2". The user in this case may use an external trigger or a "soft" trigger to initiate the output process.

Return Value:

none.

Example

```
// Create Array for SegmentList and Segments
SegmentStruct SegmentsList[2];

WORD Segment0_Data[64];
WORD Segment1_Data[64];

// Create Segment #1
for (i=0; i < (64); i++) {
    Segment0_Data[i] = ceil( 2047.0 - 2047*cos( 2*pi*i/(32) ) );
}
SegmentsList[0].SegmentNum    = 0;
SegmentsList[0].SegmentPtr    = Segment0_Data;
SegmentsList[0].NumPoints     = 64;
SegmentsList[0].NumLoops     = 0;
SegmentsList[0].BeginPadVal   = 2047;
```

```

SegmentsList[0].EndingPadVal    = 2047;
SegmentsList[0].TrigEn         = 0;
SegmentsList[0].NextSegNum     = 1;

// Create Segment #2
for (i=0; i < (64); i++) {
    Segment1_Data[i] = ceil( 2047.0 - 2047*cos( 2*pi*i/(8) ) );
}
SegmentsList[1].SegmentNum     = 1;
SegmentsList[1].SegmentPtr     = Segment1_Data;
SegmentsList[1].NumPoints      = 64;
SegmentsList[1].NumLoops       = 0;
SegmentsList[1].BeginPadVal    = 1000;
SegmentsList[1].EndingPadVal   = 1000;
SegmentsList[1].TrigEn        = 1;
SegmentsList[1].NextSegNum     = 0;    // Loops back to 1

da11000_CreateSegments(1,1,2,SegmentsList);

```

4.3.2.11 da11000_Set_Atten() [only works for DA11000 with option 1]

Description

This function call sets the amount of attenuation of the selected channel. The step size is 0.5dB. Typical insertion loss is 1.3dB. Only the first 6 bits of the "Atten_Value" are used, making the maximum amount of attenuation of 31.5dB (+ insertion loss).

Declaration

```
void da11000_Set_Atten(DWORD CardNum, DWORD ChanNum, DWORD Atten_Value)
```

Parameters

```

CardNum:    1 <= CardNum <= 4
ChanNum:    0x01, 0x02, for channels 1,2 [DA11000]
Atten_Value: 0 <= 63;

```

Return Value:

none.

Example

```
da11000_Set_Atten(1,3,30); // Sets Channel 1, Card 1, to 15dB attenuation.
```

4.3.2.12 da11000_UpdateSegmentCmds ()

Description

This function call works that same as "da11000_CreateSegments()" except that it does not download the data from system memory to card memory. Only the sequence commands are downloaded to the card's memory. This saves time when the user wants to change the order of the segments because the segment data does not have to be updated. (The micro-commands tell the memory sequencer how many times to loop, when to jump, etc.)

Declaration

```

void da11000_UpdateSegmentCmds(DWORD CardNum,
                               DWORD ChanNum,
                               DWORD NumSegments,
                               PVOID PtrToSegmentsList);

```

Parameters

CardNum: 1 <= CardNum <= 4
 ChanNum: 1 <= CardNum <= 4

NumSegments: Number of segment structures (see below) which user has defined and wants to use.

PtrToSegmentsList: Pointer to user array with each element with structure defined as shown below.

```
typedef struct
{
    DWORD    SegmentNum;    // Current Segment Number
    PVOID    SegmentPtr;   // Pointer to current user segment
                                // ==> elements of one dimensional array must
                                // be of type WORD
    DWORD    NumPoints;    // Number of points in segment
    DWORD    NumLoops;     // Number of times to repeat segment (applies
                                // to next segment)
    DWORD    BeginPadVal;  // Pad value for beginning of triggered segment
    DWORD    EndingPadVal; // Pad value for ending of triggered segment
    DWORD    TrigEn;      // If > 0 then wait for trigger before going to
                                // next segment.
    DWORD    NextSegNum;   // Next segment to jump to after completion
                                // of current segment activities
} SegmentStruct;
```

**** Note that a segment is determined to be a triggered segment by the previous segment. So setting Segment 5 as triggered will stop the sequence after Segment 5 has executed and will wait for trigger event before "NextSegNum" is started.

The first segment is a special case and is determined by default as a triggered type if SetTriggerMode() is set to "mode=2". The user in this case may use an external trigger or a "soft" trigger to initiate the output process.

Return Value:

none.

Example

See dal1000_CreateSegments() above for example.

4.4 Programming Examples**4.4.1 Using Windows 95/98/NT DLL****Example Program**

```
//
// DA11000 DLL C Example Test File
// =====
//
// 32-bit Borland C++ 5.0
//
// Web site: http://www.chase2000.cm
// Email: support@chase2000.com
//
```

```

// (C) Chase Scientific 1999
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "dal1000_dll_import.h"
#pragma link "dal1000_dll.lib"

int main(int argc, char **argv)
{
    WORD TempArray[1048575];
    int NumCards = 0;
    DWORD MemoryDepth = 1048576;
    double pi = 3.14159265358979;

    // Check to see if card available
    NumCards = dal1000_CountCards();           // Counts number of DA11000 cards

    // OPEN DRIVER
    if (NumCards > 0) then DWORD OpenErrorCode = dal1000_Open(1);    // Opens card # 1
    else exit(0);                                           // Else exits

    // INITIALIZE BOARD
    dal1000_SetTriggerMode(1,0,0);           // VERY IMPORTANT !!!

    // PUT WAVEFORM INTO ARRAY
    for (DWORD i=0; i < (MemoryDepth); i++) {
        TempArray[i] = ceil( 2047.0 - 2047*cos( 2*pi*i/(64) ) );
    }

    // CREATE SINGLE SEGMENT WITH INFINITE LOOP
    dal1000_CreateSingleSegment(1,1,MemoryDepth, 0, TempArray, 0);

    // OUTPUT DATA
    dal1000_SetTriggerMode(1,1,0);    // Enables out of data on brd# 1

    // SHUT DOWN OUTPUT
    // dal1000_SetTriggerMode(1,0,0); // Use this to shut down output on brd# 1

    // CLOSE DRIVER
    if (NumCards > 0) dal1000_Close(1); // Closes brd# 1.
}

```

Header File (for Reference)

```

//-----
#ifndef dal1000_dllH
#define dal1000_dllH
//-----

//-----
// USER ROUTINES
//-----

#define IMPORT extern "C" __declspec(dllimport)

IMPORT DWORD dal1000_CountCards(void);
IMPORT DWORD dal1000_Open(DWORD CardNum);

```

```

IMPORT DWORD dal1000_Close(DWORD CardNum);

IMPORT void dal1000_SetClock(DWORD CardNum, DWORD Frequency);

IMPORT void dal1000_SetTriggerMode(DWORD CardNum, BYTE Mode, BYTE ExtPol);
IMPORT void dal1000_SetSoftTrigger(DWORD CardNum);
IMPORT void dal1000_SetMarkers(DWORD CardNum, DWORD PointAddr, BYTE Nib1, BYTE Nib2);
IMPORT void dal1000_SetOffset(DWORD CardNum, DWORD ChanNum, int Mode, int Offset);

IMPORT void dal1000_CreateSingleSegment(DWORD CardNum, DWORD ChanNum, DWORD NumPoints,
DWORD NumLoops, PVOID UserArrayPtr, DWORD TrigEn);
IMPORT void dal1000_CreateSegments(DWORD CardNum, DWORD ChanNum, DWORD NumSegments, PVOID
PtrToSegmentsList);
IMPORT void dal1000_UpdateSegmentCmds(DWORD CardNum, DWORD ChanNum, DWORD NumSegments,
PVOID PtrToSegmentsList);
IMPORT void dal1000_Set_Atten(DWORD CardNum, DWORD ChanNum, DWORD Atten_Value);

#endif

```

5 MISCELLANEOUS

5.1 Calibration

The DA11000 has no user feature to calibrate. The gains and offsets are calibrated at the factory to be within specifications at 25°C and nominal voltages.

5.2 Maintenance

No maintenance is required. However, a yearly calibration is recommended if the user desires to maintain the DA11000 specified accuracy. Call factory for maintenance and/or extended warranty information.

5.3 Changes/Corrections to this manual

| Date | Description |
|-------------|---------------------------------|
| 01-10-2004 | First release. |
| 05-01-2009 | Exported to Open Document File. |
| | |
| | |

Trademarks:

MS-DOS, Windows 3.1, Windows 95, Windows 98, Windows ME, Windows NT, Windows 2000, and Windows XP are registered trademarks of Microsoft Corporation.